Learn Blockchains by Building One

Translation: 通过构建一个来学习区块链

The fastest way to learn how Blockchains work is to build one

Translation: 了解区块链如何工作的最快方法是构建一个

You're here because, like me, you're psyched about the rise of Cryptocurrencies. And you want to know how Blockchains work—the fundamental technology behind them.

Translation: 你来这里是因为，和我一样，你对加密货币的兴起感到兴奋。你想知道区块链是如何工作的——它们背后的基本技术。

But understanding Blockchains isn't easy—or at least wasn't for me. I trudged through dense videos, followed porous tutorials, and dealt with the amplified frustration of too few examples.

Translation: 但理解区块链并不容易——至少对我来说不是这样。我费力地浏览了密集的视频，遵循了漏洞百出的教程，并处理了太少例子带来的巨大挫折。

I like learning by doing. It forces me to deal with the subject matter at a code level, which gets it sticking. If you do the same, at the end of this guide you'll have a functioning Blockchain with a solid grasp of how they work.

Translation: 我喜欢在做中学习。它迫使我在代码级别处理主题，这让它变得棘手。如果你也这样做，在本指南的最后，你将拥有一个功能强大的区块链，并对其工作原理有着扎实的了解。

Before you get started…

Translation: 在开始之前…

Remember that a blockchain is an immutable, sequential chain of records called Blocks. They can contain transactions, files or any data you like, really. But the important thing is that they're chained together using hashes.

Translation: 请记住，区块链是一个不可变的、连续的记录链，称为区块。它们可以包含事务、文件或任何您喜欢的数据。但重要的是，它们是通过散列链接在一起的。

If you aren't sure what a hash is, here's an explanation.

Translation: 如果你不确定散列是什么，这里有一个解释。

Who is this guide aimed at? You should be comfy reading and writing some basic Python, as well as have some understanding of how HTTP requests work, since we'll be talking to our Blockchain over HTTP.

Translation: 这本指南针对的是谁？您应该能够轻松地阅读和编写一些基本的Python，并对HTTP请求的工作原理有一些了解，因为我们将通过HTTP与我们的区块链进行对话。

What do I need? Make sure that Python 3.6+ (along with pip) is installed. You'll also need to install Flask and the wonderful Requests library:

Translation: 我需要什么？确保已安装Python 3.6+（以及pip）。您还需要安装Flask和精彩的请求库：

```
pip install Flask==0.12.2 requests==2.18.4
```

Oh, you'll also need an HTTP Client, like Postman or cURL. But anything will do.

Translation: 哦，你还需要一个HTTP客户端，比如Postman或cURL。但任何事情都可以。

Where's the final code? The source code is available here.

Translation: 最后的代码在哪里？这里提供了源代码。

Step 1: Building a Blockchain

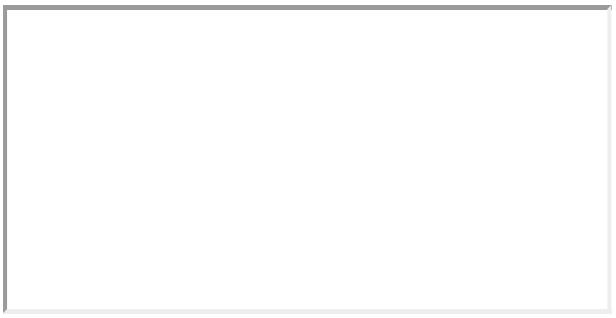
Translation: 第一步：构建区块链

Open up your favourite text editor or IDE, personally I ❤️ PyCharm. Create a new file, called blockchain.py. We'll only use a single file, but if you get lost, you can always refer to the source code.

Translation: 打开你最喜欢的文本编辑器或IDE，我个人 ❤️ PyCharm。创建一个名为blockchain.py的新文件。我们将只使用一个文件，但如果您迷路了，您可以随时参考源代码。

Representing a Blockchain

We'll create a Blockchain class whose constructor creates an initial empty list (to store our blockchain), and another to store transactions. Here's the blueprint for our class:



Translation: 我们区块链类的蓝图

Our Blockchain class is responsible for managing the chain. It will store transactions and have some helper methods for adding new blocks to the chain. Let's start fleshing out some methods.

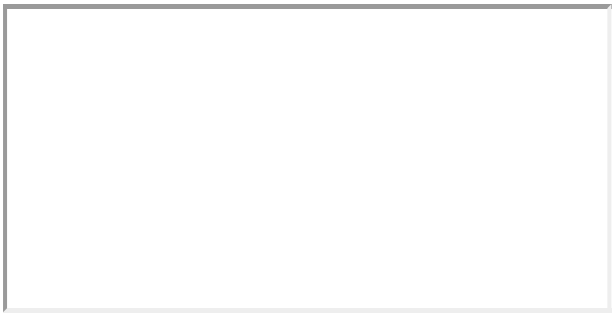
Translation: 我们的区块链类负责管理区块链。它将存储事务，并具有一些用于向链中添加新块的辅助方法。让我们开始充实一些方法。

What does a Block look like?

Translation: Block是什么样子的?

Each Block has an index, a timestamp (in Unix time), a list of transactions, a proof (more on that later), and the hash of the previous Block.

Here's an example of what a single Block looks like:



At this point, the idea of a chain should be apparent—each new block contains within itself, the hash of the previous Block. This is crucial because it's what gives blockchains immutability: If an attacker corrupted an earlier Block in the chain then all subsequent blocks will contain incorrect hashes.

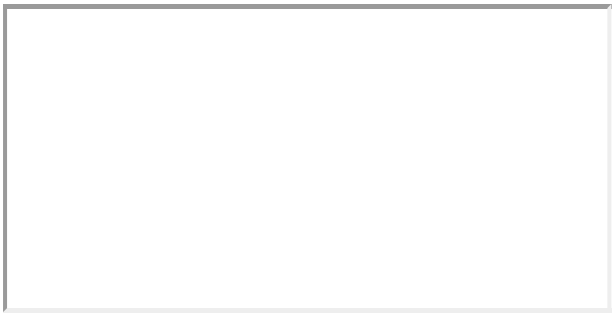
Translation: 在这一点上，链的概念应该是显而易见的——每个新块本身都包含前一个块的哈希。这一点至关重要，因为它赋予了区块链不变性：如果攻击者破坏了链中的早期块，那么所有后续块都将包含不正确的哈希。

Does this make sense? If it doesn't, take some time to let it sink in—it's the core idea behind blockchains.

Translation: 这有道理吗？如果没有，那就花点时间让它深入人心——这是区块链背后的核心理念。

Adding Transactions to a Block

We'll need a way of adding transactions to a Block. Our new_transaction() method is responsible for this, and it's pretty straight-forward:



After new_transaction() adds a transaction to the list, it returns the index of the block which the transaction will be added to—the next one to be mined. This will be useful later on, to the user submitting the transaction.

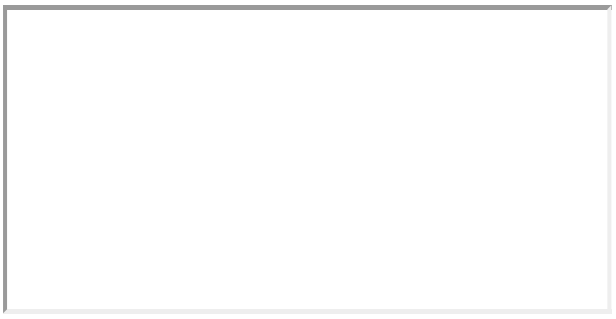
Creating new Blocks

Translation: 创建新块

When our Blockchain is instantiated we'll need to seed it with a genesis block—a block with no predecessors. We'll also need to add a "proof" to our genesis block which is the result of mining (or proof of work). We'll talk more about mining later.

Translation: 当我们的区块链被实例化时，我们需要用一个genesis块为其播种——一个没有前代的块。我们还需要为我们的成因区块添加一个"证据"，这是采矿（或工作证明）的结果。我们稍后将详细讨论采矿。

In addition to creating the genesis block in our constructor, we'll also flesh out the methods for new_block(), new_transaction() and hash():



The above should be straight-forward—I've added some comments and docstrings to help keep it clear. We're almost done with representing our blockchain. But at this point, you must be wondering how new blocks are created, forged or mined.

Understanding Proof of Work

A Proof of Work algorithm (PoW) is how new Blocks are created or mined on the blockchain. The goal of PoW is to discover a number which solves a problem. The number must be difficult to find but easy to verify—computationally speaking—by anyone on the network. This is the core idea behind Proof of Work.

Translation: 工作证明算法（PoW）是如何在区块链上创建或挖掘新块的。PoW的目标是发现一个能解决问题的数字。这个数字一定很难找到，但很容易被网络上的任何人通过计算验证。这是工作证明背后的核心思想。

We'll look at a very simple example to help this sink in.

Translation: 我们将看一个非常简单的例子来帮助理解这一点。

Let's decide that the hash of some integer x multiplied by another y must end in 0. So, hash(x * y) = ac23dc...0. And for this simplified example, let's fix x = 5. Implementing this in Python:

```
from hashlib import sha256

x = 5
y = 0  # We don't know what y should be yet...

while sha256(f'{x*y}'.encode()).hexdigest()[-1] != "0":
    y += 1

print(f'The solution is y = {y}')
```

Translation: print（f'解决方案为y= {y}'）

The solution here is y = 21. Since, the produced hash ends in 0:

Translation: 这里的解是y=21。由于，生成的哈希以0结尾：

```
hash(5 * 21) = 1253e9373e...5e3600155e860
```

In Bitcoin, the Proof of Work algorithm is called Hashcash. And it's not too different from our basic example above. It's the algorithm that miners race to solve in order to create a new block. In general, the difficulty is determined by the number of characters searched

for in a string. The miners are then rewarded for their solution by receiving a coin—in a transaction.

The network is able to easily verify their solution.

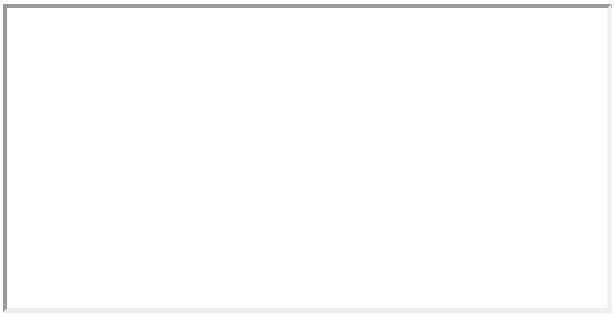
Implementing basic Proof of Work

Let's implement a similar algorithm for our blockchain. Our rule will be similar to the example above:

Translation: 让我们为我们的区块链实现一个类似的算法。我们的规则与上面的示例类似：

Find a number p that when hashed with the previous block's solution a hash with 4 leading 0s is produced.

Translation: 找到一个数字p，当与前一个块的解决方案进行哈希运算时，会产生一个以4开头的0的哈希。



To adjust the difficulty of the algorithm, we could modify the number of leading zeroes. But 4 is sufficient. You'll find out that the addition of a single leading zero makes a mammoth difference to the time required to find a solution.

Our class is almost complete and we're ready to begin interacting with it using HTTP requests.

Step 2: Our Blockchain as an API

We're going to use the Python Flask Framework. It's a micro-framework and it makes it easy to map endpoints to Python functions. This allows us talk to our blockchain over the web using HTTP requests.

Translation: 我们将使用Python Flask框架。它是一个微框架，可以很容易地将端点映射到Python函数。这允许我们使用HTTP请求通过网络与我们的区块链进行对话。

We'll create three methods:

Translation: 我们将创建三种方法：

/transactions/new to create a new transaction to a block

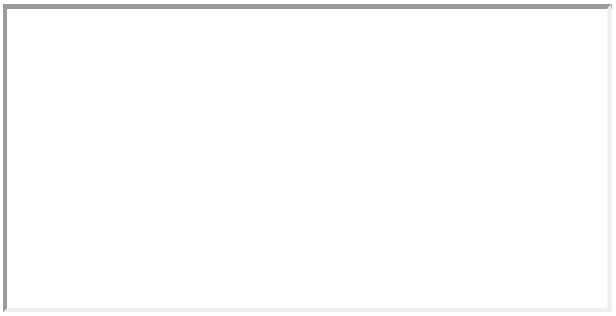
/mine to tell our server to mine a new block.

/chain to return the full Blockchain.

Setting up Flask

Our “server” will form a single node in our blockchain network. Let’s create some boilerplate code:

Translation: 我们的“服务器”将在我们的区块链网络中形成单个节点。让我们创建一些样板代码：



A brief explanation of what we’ve added above:

Line 15: Instantiates our Node. Read more about Flask here.

Line 18: Create a random name for our node.

Line 21: Instantiate our Blockchain class.

Line 24–26: Create the /mine endpoint, which is a GET request.

Line 28–30: Create the /transactions/new endpoint, which is a POST request, since we’ll be sending data to it.

Translation: 第28-30行：创建/transactions/new端点，这是一个POST请求，因为我们将向它发送数据。

Line 32–38: Create the /chain endpoint, which returns the full Blockchain.

Translation: 第32-38行：创建/chain端点，返回完整的区块链。

Line 40–41: Runs the server on port 5000.

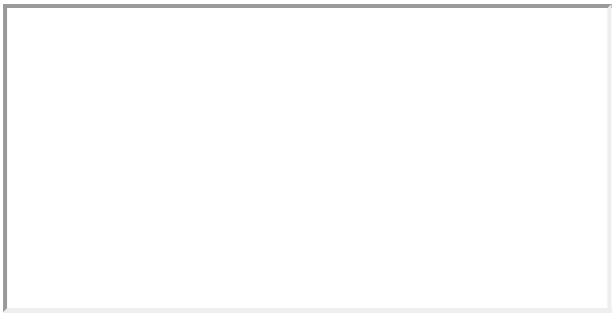
The Transactions Endpoint

This is what the request for a transaction will look like. It's what the user sends to the server:

```
{
 "sender": "my address",
 "recipient": "someone else's address",
 "amount": 5
}
```

Since we already have our class method for adding transactions to a block, the rest is easy. Let's write the function for adding transactions:

Translation: 由于我们已经有了向块中添加事务的类方法，所以剩下的就很容易了。让我们编写添加事务的函数：



Translation: 一种创建事务的方法

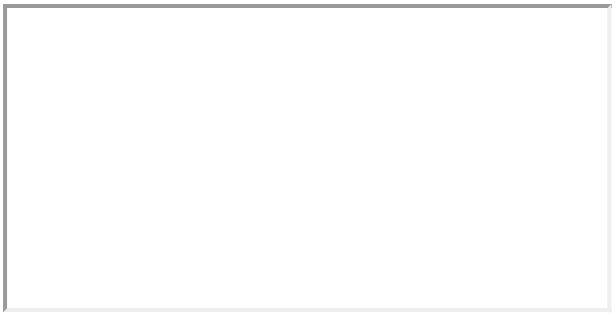
The Mining Endpoint

Our mining endpoint is where the magic happens, and it's easy. It has to do three things:

Calculate the Proof of Work

Reward the miner (us) by adding a transaction granting us 1 coin

Forge the new Block by adding it to the chain



Note that the recipient of the mined block is the address of our node. And most of what we've done here is just interact with the methods on our Blockchain class. At this point,

we're done, and can start interacting with our blockchain.

Step 3: Interacting with our Blockchain

You can use plain old cURL or Postman to interact with our API over a network.

Fire up the server:

```
$ python blockchain.py

* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

*Translation: *正在运行http://127.0.0.1:5000/（按CTRL+C退出）*

Let's try mining a block by making a GET request to http://localhost:5000/mine:

Translation: 让我们尝试通过向发出GET请求来挖掘块http://localhost:5000/mine:

Using Postman to make a GET request

Let's create a new transaction by making a POST request tohttp://localhost:5000/transactions/new with a body containing our transaction structure:

Using Postman to make a POST request

If you aren't using Postman, then you can make the equivalent request using cURL:

```
$ curl -X POST -H "Content-Type: application/json" -d '{
 "sender": "d4ee26eee15148ee92c6cd394edd974e",
 "recipient": "someone-other-address",
 "amount": 5
}' "http://localhost:5000/transactions/new"
```

Translation: $curl-X POST-H"内容类型：application/json"-d'{

I restarted my server, and mined two blocks, to give 3 in total. Let's inspect the full chain by requesting http://localhost:5000/chain:

Translation: 我重新启动服务器，挖掘了两个区块，总共得到3个。让我们通过请求检查整个链条http://localhost:5000/chain:

```
{
  "chain": [
    {
      "index": 1,
      "previous_hash": 1,
```

```
      "proof": 100,
      "timestamp": 1506280650.770839,
      "transactions": []
    },
    {
      "index": 2,
      "previous_hash": "c099bc...bfb7",
      "proof": 35293,
      "timestamp": 1506280664.717925,
      "transactions": [
        {
          "amount": 1,
          "recipient": "8bbcb347e0634905b0cac7955bae152b",
          "sender": "0"
        }
      ]
    },
    {
      "index": 3,
      "previous_hash": "eff91a...10f2",
      "proof": 35089,
      "timestamp": 1506280666.1086972,
      "transactions": [
        {
          "amount": 1,
          "recipient": "8bbcb347e0634905b0cac7955bae152b",
          "sender": "0"
        }
      ]
    }
  ],
  "length": 3
}
```


Step 4: Consensus

This is very cool. We've got a basic Blockchain that accepts transactions and allows us to mine new Blocks. But the whole point of Blockchains is that they should be decentralized. And if they're decentralized, how on earth do we ensure that they all reflect the same chain? This is called the problem of Consensus, and we'll have to implement a Consensus Algorithm if we want more than one node in our network.

Registering new Nodes

Before we can implement a Consensus Algorithm, we need a way to let a node know about neighbouring nodes on the network. Each node on our network should keep a registry of other nodes on the network. Thus, we'll need some more endpoints:

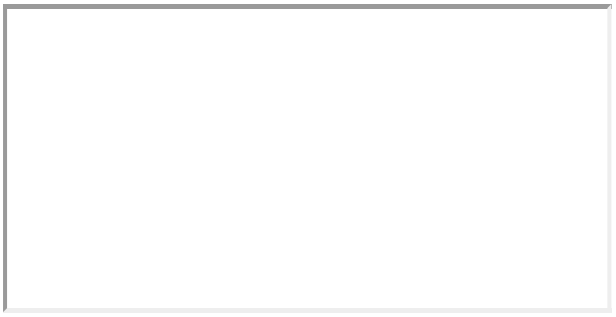
Translation: 在我们实现共识算法之前，我们需要一种方法让节点知道网络上的相邻节点。我们网络上的每个节点都应该保存网络上其他节点的注册表。因此，我们需要更多的端点：

/nodes/register to accept a list of new nodes in the form of URLs.

Translation: /节点/注册以接受URL形式的新节点列表。

/nodes/resolve to implement our Consensus Algorithm, which resolves any conflicts—to ensure a node has the correct chain.

We'll need to modify our Blockchain's constructor and provide a method for registering nodes:



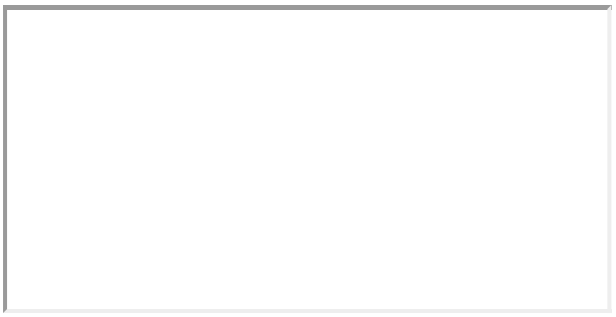
Note that we've used a set() to hold the list of nodes. This is a cheap way of ensuring that the addition of new nodes is idempotent—meaning that no matter how many times we add a specific node, it appears exactly once.

Implementing the Consensus Algorithm

Translation: 实现共识算法

As mentioned, a conflict is when one node has a different chain to another node. To resolve this, we'll make the rule that the longest valid chain is authoritative. In other words, the longest chain on the network is the de-facto one. Using this algorithm, we reach Consensus amongst the nodes in our network.

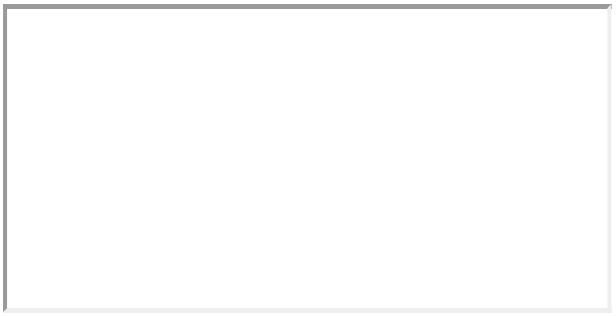
Translation: 如前所述，冲突是指一个节点与另一个节点具有不同的链。为了解决这个问题，我们将制定最长有效链具有权威性的规则。换句话说，网络上最长的链就是事实上的链。使用该算法，我们可以在网络中的节点之间达成共识。



The first method valid_chain() is responsible for checking if a chain is valid by looping through each block and verifying both the hash and the proof.

resolve_conflicts() is a method which loops through all our neighbouring nodes, downloads their chains and verifies them using the above method. If a valid chain is found, whose length is greater than ours, we replace ours.

Let's register the two endpoints to our API, one for adding neighbouring nodes and the another for resolving conflicts:



At this point you can grab a different machine if you like, and spin up different nodes on your network. Or spin up processes using different ports on the same machine. I spun up another node on my machine, on a different port, and registered it with my current node. Thus, I have two nodes: http://localhost:5000 and http://localhost:5001.

Translation: 在这一点上，如果你愿意，你可以抓取一台不同的机器，并在网络上旋转不同的节点。或者在同一台机器上使用不同端口启动进程。我在机器上的另一个端口上启动了另一个节点，并将其注册到当前节点。因此，我有两个节点：http://localhost:5000和 http://localhost:5001.

Registering a new Node

Translation: 注册新节点

I then mined some new Blocks on node 2, to ensure the chain was longer. Afterward, I called GET /nodes/resolve on node 1, where the chain was replaced by the Consensus Algorithm:

And that's a wrap... Go get some friends together to help test out your Blockchain.

I hope that this has inspired you to create something new. I'm ecstatic about Cryptocurrencies because I believe that Blockchains will rapidly change the way we think about economies, governments and record-keeping.

Translation: 我希望这能激励你去创造一些新的东西。我对加密货币感到欣喜若狂，因为我相信区块链将迅速改变我们对经济、政府和记录的看法。

Update: I'm planning on following up with a Part 2, where we'll extend our Blockchain to have a Transaction Validation Mechanism as well as discuss some ways in which you can productionize your Blockchain.

Translation: 更新：我计划在第2部分中跟进，我们将扩展我们的区块链，使其具有交易验证机制，并讨论您可以将区块链产品化的一些方法。

If you enjoyed this guide, or have any suggestions or questions, let me know in the comments. And if you've spotted any errors, feel free to contribute to the code here!

Translation: 如果你喜欢这本指南，或者有任何建议或问题，请在评论中告诉我。如果你发现了任何错误，请随时在这里贡献代码！